Pergamon

Computers Math. Applic. Vol. 35, No. 8, pp. 9–14, 1998

0898-1221/98 $19.00 + 0.00

PII: S0898-1221(98)00041-8

A Fast and Robust RNS Algorithm for Evaluating Signs of Determinants

V. S. DIMITROV, G. A. JULLIEN AND W. C. MILLER
VLSI Research Group, University of Windsor
401 Sunset Avenue, Windsor, Ontario, Canada N9B 3P4

(Received September 1997; accepted October 1997)

Abstract—A new and efficient number theoretic algorithm for evaluating signs of determinants is proposed. The algorithm uses computations over small finite rings. It is devoted to a variety of computational geometry problems, where the necessity of evaluating signs of determinants of small matrices often arises.

Keywords—Algorithms, Residue number systems, Computational geometry.

1. INTRODUCTION

Recent work has focused on performing efficient single precision arithmetic computation of signs of determinants. The problem has arisen in many computational geometry algorithms [1–6], for example, to determine if a point belongs to a given half-space or a given ball; it is very important to have fast and robust algorithms to perform such tests.

The numerical stability of this problem requires special attention, and is specifically addressed in this new technique. For the case of 2×2 matrices, the problem can be stated as follows.

Let $D = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix}$ be a 2×2 determinant whose entries are b-bit integers; evaluate the sign of D using only b-bit arithmetic.

The previous results [1,7] on this subject have been obtained by the use of some simple geometric considerations; specifically:

(1) the entries of the matrix are integers, and
(2) we are looking for a single-precision arithmetic solution.

Therefore, it is natural to try to find an efficient algorithm based on a number system which is particularly suited to work with integers. Residue number systems (RNS) [8] fall into this category. The most visible advantage of this class of number systems is their ability to perform addition, subtraction, and multiplication in a parallel, carry-free manner using independent computational channels for calculations over small finite rings. Division within this system has been recognized as a difficult and time-consuming operation [9,10]; however, for matrices of small size (i.e., 2×2 or 3×3), we need only perform addition, subtraction, and multiplication. Therefore, one can very easily compute the residue format of the determinant. The final step, namely, sign detection, is the most difficult task. Here we will use a routine from the recently proposed algorithm by Hitz and Kaltofen [9].

The asymptotic complexity of our algorithm is lower than that of previously published algorithms, and we can show this using straightforward analytic number theory considerations. However, what is more important is that the algorithm can be used in massive computational

geometry problems such as computations of Voronoi diagrams for the case of a large number of 'generators' (points), in some extreme cases more than one million [5]. Since our algorithm consists almost entirely of look-up table operations on fairly small look-up tables, it is very well suited for VLSI implementation. The computational example presented at the end of this paper clarifies the idea and highlights some of the opportunities for further improvements.

A very interesting feature of our algorithm is the fact that it detects negative determinants usually much faster than positive ones, and this can be useful in some particular cases.

2. DEFINITIONS

We shall use the same notation as in [9].

Let $m'_1, m'_2, \ldots, m'_{2n} \in Z$, $1 < m'_1 < m'_2 < \cdots < m'_{2n-1} < m'_{2n}$ be a set of $2n$ small, possibly consecutive primes (we will clarify the term 'small' later). It is enough to choose these numbers to be only pairwise relatively prime, but it complicates the complexity analysis without improving the asymptotic performance of the algorithm.

Let us group these $2n$ moduli into two vectors of size n:

$$m_1 = m'_1,\ m_2 = m'_3, \ldots, m_n = m'_{2n-1} \qquad \text{and} \qquad m_{n+1} = m'_2,\ m_{n+2} = m'_4,\ m_{2n} = m'_{2n}.$$

The RNS defined by these moduli is called an *extended* RNS [9]; the system defined by the first n moduli is the *base* RNS, and the RNS defined by the remaining moduli is the *extension* RNS. The respective *dynamic range* for the base and the extension RNS is:

$$M = \prod_{i=1}^{n} m_i \qquad \text{and} \qquad \overline{M} = \prod_{i=n+1}^{2n} m_i,$$

where the range for the entire extended RNS is $M\overline{M}$. An integer X, $0 \le X < M\overline{M}$ with residues $x_1 = X \bmod m_1, \ldots, x_{2n} = X \bmod m_{2n}$, will be represented in the extended RNS by $(x_1, \ldots, x_n; x_{n+1}, \ldots, x_{2n})$.

The representation in the associated *mixed radix system* (MRS) will be denoted by

$$\langle v_1, \ldots, v_n; v_{n+1}, \ldots, v_{2n} \rangle,$$

where

$$X = \sum_{i=1}^{2n} v_i P_{i-1}, \quad P_0 = 1, \quad P_i = \prod_{j=1}^{i} m_j, \qquad \text{for } 1 \le i < 2n$$

and $0 \le v_i < m_i$, for $1 \le i \le 2n$. Hitz and Kaltofen [9] mention several important properties of the extended RNS. For our purposes, i.e., evaluating signs of determinants, the most important property is that the multiplication of two base RNS numbers in the extended RNS will never produce an overflow because $M^2 < M\overline{M}$.

3. THE ALGORITHM FOR 2×2 MATRICES

Given a determinant $D = \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \end{bmatrix} = x_1 y_2 - y_1 x_2$, where x_1, x_2, y_1, y_b are b-bit integers, we want to evaluate the sign of D using only b-bit arithmetic. Of course, the direct computation may cause overflow.

Let us select the prime moduli m_i of the base RNS with the proviso that:

$$M = \prod_{i=1}^{n} m_i \ge 2^{2b}. \tag{1}$$

It follows from the Prime Number Theorem [11] that $n = O(b/\log b)$. The definition of the extended RNS guarantees that $\overline{M} > M \ge 2^{2b}$. Therefore, the total dynamic range covered by the extended RNS is at least $4b$-bits.

3.1. Description of the Algorithm

We propose the following algorithm.

Input: x_1, x_2, y_1, y_2: b-bit integers;

Output: the sign of $x_1y_2 - x_2y_1$.

Step 1. Represent the entries in extended RNS (using the dynamic range, M, defined in equation (1)):

$$x_1 \to (x_1 \bmod m_1, x_1 \bmod m_2, \ldots, x_1 \bmod m_{2n}) = \left(x_1^{(1)}, x_1^{(2)}, \ldots, x_1^{(2n)}\right),$$

and similarly for x_2, y_1, and y_2.

Step 2. Compute the vector:

$$\mathbf{z} = \left(\underbrace{z^{(1)}, z^{(2)}, \ldots, z^{(n)}}_{\text{base residues}}, \underbrace{z^{(n+1)}, z^{(n+2)}, \ldots, z^{(2n)}}_{\text{extension residues}}\right),$$

where $z^{(i)} = (x_1^{(i)}y_2^{(i)} - x_2^{(i)}y_1^{(i)}) \bmod m_i$.

Step 3. Perform the base extension for the vector $\mathbf{z}^* = (z^{(1)}, z^{(2)}, \ldots, z^{(n)})$. Let the result obtained be: $\mathbf{z}^*_{(e)} = (z^{(1)}, z^{(2)}, \ldots, z^{(n)}, z_{(e)}^{(n+1)}, z_{(e)}^{(n+2)}, \ldots, z_{(e)}^{(2n)})$.

Step 4. Test the vectors $(z^{(n+1)}, z^{(n+2)}, \ldots, z^{(2n)})$ and $(z_{(e)}^{(n+1)}, z_{(e)}^{(n+2)}, \ldots, z_{(e)}^{(2n)})$ for equality. If all components agree, then output '*nonnegative determinant*'; otherwise output '*negative determinant*'. If at least one of the components of $\mathbf{z}$ is nonzero, then the '*nonnegative determinant*' is '*positive*'; otherwise it is *zero*.

3.2. Proof of Correctness

We want to compare the numbers x_1y_2 and x_2y_1, where they are strictly smaller than 2^{2b}. If the result of the subtraction (the value of the determinant) is positive, this maps to the range $[0, M)$ and the extension residues from Step 2 will be identical to the extension residues from Step 3 since the number is uniquely represented by $\mathbf{z}^*$. On the other hand, if the subtraction is negative, then this result maps to the range $(M, M\overline{M})$ in the extended RNS representation. In this case, $\mathbf{z}^*$ does not uniquely represent the result, and the extension residues from Steps 2 and 3 will no longer be equal.

3.3. Complexity Analysis

Now we analyze the complexity of the algorithm.

Step 1 requires $O(b/\log b)$ divisions to obtain the residues. Note that this is the only step where we need b-bit arithmetic.

Step 2 requires $O(b/\log b)$ operations to calculate the components of $\mathbf{z}$.

Step 3 is the most important part of the algorithm, and it deserves special attention. The conversion from base RNS to the extension RNS needs the following computational operations:

(1) given $\mathbf{z}^* = (z^{(1)}, z^{(2)}, \ldots, z^{(n)})$, compute $Z^* = \sum_{i=1}^{n} v_i P_{i-1}$;

(2) evaluate, in succession, for every modulus m_j with $n+1 \le j \le 2n$:

$$Z^* \bmod m_j = \left(\sum_{i=1}^{n} v_i P_{i-1}\right) \bmod m_j = \left(\sum_{i=1}^{n} (v_i \bmod m_j)(P_{i-1} \bmod m_j)\right) \bmod m_j. \qquad (2)$$

The constants $P_{i-1} \bmod m_j$ can be precomputed and kept in a look-up table, while the products v_iP_{i-1} may be computed in parallel [9] on $n \times n$ RNS processor elements. Finally, the summation $\bmod m_j$ can be performed in binary trees with $O(\log n) = O(\log\log b)$ depth. The total runtime estimation of this step is $O(b \log\log b/\log b)$ operations, and this determines the asymptotic complexity of the algorithm.

Step 4 requires at most $O(b/\log b)$ comparisons.

In summarizing, we may estimate that the overall runtime is $O(b \log\log b/\log b)$ operations.

3.4. A Numerical Example

We exemplify the procedure with one numerical example.

Consider the case $b = 8$, that is, the entries are 8-bit integers. Let us choose the matrices:

$$A_1 = \begin{bmatrix} 255 & 250 \\ 249 & 244 \end{bmatrix} \quad \text{and} \quad A_2 = \begin{bmatrix} 250 & 255 \\ 244 & 249 \end{bmatrix}.$$

Obviously, their determinants have opposite signs.

For this value of b, we may select the set of moduli given in Table 1.

Therefore, we work with $n = 4$ base moduli, namely, $(m_1, m_2, m_3, m_4) = (11, 17, 23, 31)$ and four extension moduli $(m_5, m_6, m_7, m_8) = (13, 19, 29, 37)$.

Table 1. Extended RNS moduli for the case $b = 8$.

m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8
11	17	23	31	13	19	29	37

The dynamic range for the base RNS is $M = 133331$, and for the extension RNS $\overline{M} = 265031$.

The total dynamic range of the extended RNS is $M\overline{M} = 35336848261 \approx 2^{35.04}$.

The values of $P_i \bmod m_j$, $i = 0, 1, 2, 3$; $j = 5, 6, 7, 8$ are precomputed and stored as given in Table 2.

Table 2. Precomputed values of $P_i \bmod m_j$.

	mod 13	mod 19	mod 29	mod 31
$P_0 = 1$	1	1	1	1
$P_1 = 11$	11	11	11	11
$P_2 = 187$	5	16	13	2
$P_3 = 4301$	11	7	9	9

The final preprocessing step we need is the determination of the coefficients v_i, necessary in Step 3 of the algorithm (i.e., equation (2)). In the case of the above moduli, these coefficients can be evaluated as follows:

$$\begin{aligned} v_1 &= z_1 \bmod m_1, \\ v_2 &= 14(z_2 - v_1) \bmod m_2, \\ v_3 &= 8(z_3 - v_1 - 11v_2) \bmod m_3, \\ v_4 &= 27(z_4 - v_1 - 11v_2 - v_3) \bmod m_4. \end{aligned}$$

Now let us consider the application of our algorithm for the matrix A_1.

Step 1 transforms this matrix into an extended RNS format:

$$A_1 \to \begin{bmatrix} (2,0,2,7,8,8,23,33) & (8,12,20,2,3,3,18,28) \\ (7,11,19,1,2,2,17,27) & (2,6,14,27,10,16,12,20) \end{bmatrix}.$$

Step 2 computes the residue format of the determinant of A_1:

$$\det(A_1) \to (3,4,16,1,9,8,28,7).$$

Neither of these steps requires communication between different computational channels and can be performed in parallel.

Step 3 must evaluate the sign of the number obtained in Step 2. To do this, we select the first four residues, namely, $(z_1, z_2, z_3, z_4) = (3, 4, 16, 1)$, compute the corresponding v_1, v_2, v_3, and v_4,

and extend the residues of this number modulo m_5, m_6, m_7, and m_8. We can express this in pseudo code as follows:

```
for(j=5;j<= 8;j++) {
    s=0; for(i=0;i<=3;i++){
    g=((v[i+1] % m[j]) * P[i][j]) % m[j]; s+=g;s%=m[j];}
if(s!=z[j]) return('negative determinant');}
return('positive determinant');
```

The computed residues modulo m_5, m_6, m_7, and m_8 form a vector $(12, 16, 17, 27)$, which is different from $(9, 8, 28, 7)$; therefore, the determinant is negative. Its true value (-30) remains unknown at the end of the algorithm. In fact, the algorithm terminates immediately after determining that $Z_5 \neq 9$.

In the case of matrix $A_2 = \begin{bmatrix} 250 & 255 \\ 244 & 249 \end{bmatrix}$, the residue format of the determinant is

$$\det(A_2) \rightarrow (8, 13, 7, 30, 4, 11, 1, 30).$$

Starting with the vector $(8, 13, 7, 30)$ and repeating the same procedure, we find residues $(4, 11, 1, 30)$ modulo m_5, m_6, m_7, and m_8; equality with the last four components of the residue format of $\det(A_2)$ indicates a positive determinant.

This example reveals one of the main features of the algorithm. To detect negative determinants, one needs to find at least one modulus (say m_k), for which the obtained value of the residue z_k will disagree with the corresponding s (obtained in the main cycle of the above procedure). For example, if the determinant is negative, we may expect that in about 92.3 percent of the cases (12/13), it will be detected on the first test, namely, a test that uses $m_5 = 13$. Therefore, the above procedure can be slightly improved if we change the main cycle to `for(j=8;j>=5;j--)`. In this case, we start the testing with the largest modulus (in this example, 37), and this increases the chance to detect a negative determinant on the first test to 97.3 percent (36/37).

If the determinant is positive, however, we have to perform the complete test to make sure that all of the extension residues match. The algorithm therefore seems to be particularly effective if one has to check the signs of many determinants and knows in advance that the majority of the determinants will be negative.

What about negative entries? The problem is easily solved. If x_1y_2 and x_2y_1 have different signs, we have nothing to do; otherwise we compare $a = |x_1||y_2|$ and $b = |x_2||y_1|$.

3.5. 3×3 Matrices

In this case, we may apply the same method. We will mention two things.

1. Now the moduli must be chosen to satisfy the condition $\prod_{i=1}^{n} m_i \geq 5 \cdot 2^{3b}$, which is a sufficient guarantee against overflow. This can be shown as follows. Let

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

be a 3×3 matrix with b-bit integer entries. Now we have six summands of the form $\varepsilon a_{1j_1} a_{2j_2} a_{3j_3}$, where $\varepsilon = \pm 1$ and (j_1, j_2, j_3) are all permutations of the set $\{1, 2, 3\}$. Since the summands can neither be all positive or all negative, the worst case occurs when the first five of them have, say, a positive sign, and the sixth has a negative sign. If the moduli satisfy the condition $\prod_{i=1}^{n} m_i \geq 5 \cdot 2^{3b}$, then the exactness of the algorithm is guaranteed for this worst case scenario.

In some applications, this condition might be unnecessarily pessimistic. Let us consider the following problem.

Given a set of three points in the Euclidean plane $A = (x_1, y_1)$, $B = (x_2, y_2)$, and $C = (x_3, y_3)$; determine if this particular order, namely, (A, B, C), forms a clockwise or counter-clockwise triple. Clearly, this is equivalent to finding the sign of the determinant

$$\Delta = \begin{bmatrix} 1 & 1 & 1 \\ x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \end{bmatrix}.$$

The existence of ones in the first row relaxes the dynamic range condition to $\prod_{i=1}^{n} m_i \geq 5 \cdot 2^{2b}$.

2. The computational complexity has the same asymptotic performance as in the case of matrices of size 2×2 (i.e., $O(b \log \log b / \log b)$).

4. CONCLUSIONS

In this paper, we have presented a new algorithm for evaluating of the sign of determinants for matrices of small size; this problem frequently arises in many computational geometry algorithms. Our approach is based on a residue number system computation, which combines the advantages of parallelism and modularity. The only required operations are additions, subtractions, and multiplications over very small finite rings, which makes possible their implementation using look-up tables. The technique is particularly suitable fro VLSI hardware implementation, where one may exploit the independent computational power of RNS arithmetic.

REFERENCES

1. K.L. Klarkson, Safe and effective determinant evaluation, In *Proc. 33rd Annual IEEE Symp. Found. Comput. Sci.*, pp. 387–395, (1992).
2. S. Fortune and V. Milenkovic, Numerical stability of algorithms for line arrangements, In *Proc. 7th Annual ACM Symp. Comput. Geom.*, pp. 334–342, (1991).
3. S. Fortune and C.J. Van Wyk, Efficient exact arithmetic for computational geometry, In *Proc. 9th Annual Symp. Comput. Geom.*, pp. 163–172, (1993).
4. M. Karasik, D. Lieber and L.R. Nackman, Efficient Delaunay triangulations using rational arithmetic, *ACM Trans. Graph.* **10**, 71–91 (1991).
5. K. Sugihara and M. Iri, Construction of the Voronoi diagram for 'one million' generators in single-precision arithmetics, In *Proc. of the IEEE,* Special Issue on *Comput. Geom.* **80**, 1471–1484 (1992).
6. K. Sugihara and M. Iri, Two design principles of geometric algorithms in finite precision arithmetic, *Appl. Math. Lett.* **2** (2), 203–206 (1989).
7. F. Avnaim, J.-D. Boissonnat, O. Devillers, F.R. Preparata and M. Yvinec, Evaluating signs of determinants using single-precision arithmetic, *Algorithmica* **11**, 111–132 (1997).
8. M.A. Soderstrand, W.K. Jenkins, G.A. Jullien and F.J. Taylor, *Residue Number System Arithmetic: Modern Applications in Digital Signal Processing*, IEEE Press, (1986).
9. M. Hitz and E. Kaltofen, Integer division in residue number systems, *IEEE Trans. Computers* **44**, 983–989 (1995).
10. W.A. Chren, Jr., A new residue number system division algorithm, *Computers Math. Applic.* **19** (7), 13–29 (1990).
11. E. Bach, *Analytic Methods in the Analysis and the Design of Number-Theoretic Algorithms*, MIT Press, (1985).